Project Title: SUDOKU QUEST

Team members:

Lavesh Kumar Srivastav (145320004)

Sayandeep Roy (145060019)

Dipayan Mukherjee (145320014)

Project Description:

❖ Aim of Project:

The game Sudoku, since its first appearance in 1970's, has been one of the most popular puzzle games of all times. It is often seen that some people have a real knack of solving the puzzle easily, while there are others who face difficulties at first and require more and more practice to acquire that skill.

We have set the goals of our project to aid those who are immensely fond of the game but with amateur skills. The program will work both as a ***Sudoku generator*** by providing solvable Sudoku puzzles for user to play and practice (according to user specified difficulty levels) and also as an intelligent ***Sudoku auto-solver***, which can help the user play the game by providing appropriate 'hints' and 'check' options during play or show the end solution of puzzle if user wants. User will also have the option to give any unsolved puzzle as input to the program and check its solution.

❖ Project Ideas:

The project involves basically three broad components:

1) Solving an user defined puzzle taken as inputs,
2) Generating a puzzle for the user to play,
3) Building the graphics and interfaces of the game.

➢ Solving a Sudoku puzzle:

The Algorithm that we have chosen to solve a random Sudoku puzzle is called the ***Back-tracking algorithm***. The logic behind this is to basically iterate all the possible solutions for a given unassigned cell and assign them one by one until the

Sudoku is solved and 'back-track' whenever required. The summary of steps are given below:

- First in the given Sudoku puzzle, we have to locate an unassigned (empty) cell.
- Then numbers from 1 to 9 are successively checked until a 'valid' number is found i.e. the number should be unique in that row, column and box (3X3) to which the cell belongs.
- This valid number is then assigned to that cell.
- Next, another unassigned location is sought for and this process is repeated to fill that up also.
- The 'back-tracking' comes to action if all the possible numbers from 1 to 9 are invalid for any cell that is currently being checked. The algorithm then moves back to the previous cell and changes that cell's value to another valid number, i.e. back tracks. Afterwards, it moves again to the next cell and the whole process repeats itself.
- A Sudoku will be considered unsolvable if back tracking has been done up to the first filled cell and all digits have been tried and nothing worked even at that cell.

(**Reference:** http://see.stanford.edu/materials/icspacs106b/H19RecBacktrackExamples.pdf)

➢ Generating a Sudoku puzzle:

The logic behind the generation procedure of a Sudoku is that we can start with an empty Sudoku grid.

- We can then start choosing a random cell in that grid and place a random value ranging from 1 to 9.
- Before choosing each random location we need to check if the location is empty or not. Also in an empty cells, before putting the random value, we need to check if it is valid as per rules of Sudoku.
- It is a proven fact that a Sudoku puzzle having less than 17 given entries cannot have a unique solution. So, to generate a Sudoku puzzle we are bound to give at least 17 entries as a **Sudoku with multiple solution is not considered as a valid puzzle.**
- But, giving more than 16 entries do not guarantee a puzzle to have a unique solution. It is found that Sudoku puzzle with maximum 77 number of given entries can also have multiple solutions. So after having 16 entries, from the 17th entry onwards we will have to check whether the Sudoku will have a unique solution. If yes, then only we should choose and fill up the next random cell and repeat the process of checking the uniqueness of the solution. If no, then the next random value is assigned to that cell.

- Thus in this process we can fill up certain no. of locations in a Sudoku grid according to difficulty level chosen by the user.

❖ Project Execution:

The project is executed in three parts, in co-ordination with the three components of the program.

➢ Header files to be included:

1) gtk/gtk.h; **2)** string.h; **3)** iostream; **4)** stdlib.h; **5)** stdio.h; **6)** gdk/gdk.h; **7)** time.h **8)** cstdio

➢ Building the Sudoku Auto-solver program:

In C++ environment, the Sudoku auto-solver program is to be written and for that the list of functions and the purpose they are to serve, are described below:

▪ ‘invalid_entry’ Function:

- Type: bool

This function will check whether the Sudoku given by the user to solve, actually is a valid one or not. The procedures include checking whether the entered Sudoku has unique numbers in rows, columns and boxes.

▪ ‘isUnique’ Function:

- Type: bool

This function will check whether the user specified Sudoku puzzle is actually having a unique solution or not. If it has more than one solution then it will return false after displaying appropriate message. If it is a valid Sudoku then the function will return true and proceed. The purpose and work method of the function are discussed below:

1) Given an unsolved Sudoku puzzle, first the unsolved grid is saved in two different places. These two same unsolved grid are then solved twice.
2) At first, taking one unsolved grid the trial numbers in an empty cell are put starting from 1 to 9. After the Sudoku is completely solved, the whole solved Sudoku is stored somewhere.
3) After that, the other unsolved Sudoku grid is solved by the same method, but this time the trial numbers are put starting from 9 and decreased to end at 1.
4) If the two solutions are equal, then it is confirmed that the given unsolved Sudoku puzzle has unique solution.

- <u>**'solveSudoku' Function:**</u>
    - <u>Type:</u> bool

This is the main function that will be called whenever a Sudoku puzzle is needed to be solved. All the other functions related to solving the Sudoku are called under this function. The purpose and work method of the function are discussed below:

1) Get the user specified unsolved Sudoku grid (a 9X9 2-D arrary) as input.
2) Find an empty box in the grid using the **findUnassignedLocation** function, whose row and column numbers are stored in two variables 'row' and 'col'.
3) Whenever an empty place is found, use an iteration to start putting trial numbers starting from 1 to 9 in that place and check if it abides by the rule of the game. The process of checking consists of the following steps:
    i) Check if the corresponding row contains that number using the **checkRow** function. If YES, then change the trial number to the next number. If NO, then proceed.
    ii) Check if the corresponding column contains that number using the **checkColumn** function. If YES, then change the trial number to the next number. If NO, then proceed.
    iii) Check if the corresponding 3X3 box contains that number using the **checkBox** function. If YES, then change the trial number to the next number. If NO, then proceed.
4) After a valid number is found, the number is put in that location.
5) Then this **solveSudoku** function is called **recursively** again and all the steps are again done to find and fill up another unassigned location in the grid.
6) Continuing this process, if a time comes when the program will find an empty place, where no valid number can be put then any trial number put in the place will be removed making it unassigned again and the function will return false and return back to the previous function that called it.
7) It then goes back to the previous empty place in that function which was filled by a number and replaces the number with the next possible number and again recursive call is made. **Thus the back tracking method is implemented efficiently by this function**.
8) If all the cells are filled up then the grid is complete and the **solveSudoku** will then return true.

- **‘findUnassignedLocation’ Function:**
    - Type: bool

The function is designed to find an unfilled cell in the given 2-D array (Sudoku grid). The function takes input the user defined grid and the ‘row’ and ‘col’ variable declared under solveSudoku function by references. The function returns true when an empty cell is found, after assigning the row no. and column no. of that location to the respective variables ‘row’ and ‘col’. If no unassigned location is found then it will return false and go back to the caller solveSudoku function.

- **‘isSafetoPut’ Function:**
    - Type: bool

The function will take the grid, row and column number of the unassigned location and the trial number to be put in that location as input and check whether the number is valid in that location or not, by calling the **checkRow**, **checkColumn**, **checkBox** functions. If any one of the three returns false then isSafetoPut function will return false and go back to the **solveSudoku** function.

- **‘checkRow’ Function:**
    - Type: bool

The function will check whether the trial number is unique in that row and return true if it is and false if the same number is already present in that row.

- **‘checkColumn’ Function:**
    - Type: bool

The function will check whether the trial number is unique in that column and return true if it is and false if the same number is already present in that column.

- **‘checkBox’ Function:**
    - Type: bool

The function will check whether the trial number is unique in that 3X3 box and return true if it is and false if the same number is already present in that box.

➢ Building the Sudoku Generator program:

▪ ‘sudoku_game’ Function:

- Type: Callback function of type void.

The function performs the steps for generating a valid Sudoku puzzle for the user to play. The steps involved are written below:

1) Four difficulty levels, **‘Beginner’**, **‘Amateur’**, **‘Expert’**, and **‘Veteran’** were assigned for the user to choose according to his/her preferences.

2) File handling system was involved to store ten valid Sudoku puzzles for each of the difficulty levels. In the .txt file we stored the numbers in such a way that for a single difficulty level, e.g. Beginner level there are a total of 810 numbers, where each set of 81 numbers representing a valid Sudoku. The blank spaces are denoted by the number ‘0’. Each set is separated by ‘|’ symbol and thus for one difficulty level, there are 10 such sets. Now for the next difficulty level, say ‘Amateur’, there are also ten such sets of numbers and the last set of numbers in beginner level being separated from the 1st set of Amateur level by the symbol ‘@’. Same thing is done for the rest of the difficulty levels as well in that ordered manner.

Here is a simple layout:

Beginner: 1st set |Beginner: 2nd set| Beginner: 3rd set|…..Beginner: 10th set @ Amateur: 1st set| Amateur: 2nd set| Amateur: 3rd set|…….Amateur: 10th set @ Expert: 1st set…………………..Veteran: 10th set

Total no of characters: 3240 digits + 360 “|” s + 3 “@” s = 3603
“@” s are at the positions: 820th , 1640th , 2460th positions.

3) Next, in the generator function, a random number is chosen by using **‘rand’** function. For beginner level, the number is chosen from 0 to 9, with each representing one set of 81 numbers. For example, if the random number came out to be 3, then the 81 numbers, starting from the position (3X82) or 246th to (246+81)th position are taken to be displayed in the grid.

4) Now according to the particular set chosen, the numbers from that set are taken up and grid is filled with the numbers and Sudoku is generated. The empty boxes are filled with zero as usual. The Sudoku is displayed on the screen for the user to play.

▪ ‘hint’ Function:

- Type: Callback function of type void.

The purpose of this function is to help the user to play the game by filling any one random cell with a number that is appropriate to solve the puzzle and have a unique solution. The grid provided by the user is first copied into a 9X9 'testGrid' array. This array is then passed through the 'solveSudoku' function in order to get a filled array. Then an empty box is chosen in the main grid where the value from the corresponding box of testGrid was put.

- **'check1' Function:**
    - Type: bool.

This function will enable the user to check whether the entries given by him/her so far are correct or not. The function basically copies the Sudoku grid which is to be checked, in a 9X9 'testGrid' array. Then the latter is passed in the 'solveSudoku' function and a completed puzzle is obtained. Then the values of the non-empty boxes of the main grid are compared with the corresponding boxes in the solved 'testGrid' array. If the numbers match, then appropriate message is displayed to the user that he is on the right track of solving the game. If not, then also a warning message will appear.

➢ Building the Graphics and interfaces:

GTK+, or the GIMP Toolkit, is a multi-platform toolkit for creating graphical user interfaces. Offering a complete set of widgets, GTK+ is suitable for projects ranging from small one-off tools to complete application suites. We will be using the C++ friendly GTK+, which is a free software and part of the GNU Project.

- On executing the program, user will be presented with a window containing ***"Welcome "message*** and three buttons:

(i) ***Sudoku Game*** :
On pressing this button, new window pops up displaying a 9x9 Sudoku grid in which some cells contain numbers.
Menu bar with options:
File: 1.***New*** 2. ***Exit***
Help: 1.***Check*** 2.***Hint*** 3.***About***

(ii) ***Sudoku Solver***:
On pressing this button, new window with empty 9x9 grid is presented. User have to enter the unsolved Sudoku. At the bottom of window there is "***Solve the Sudoku***" button which solves the Sudoku and displays the solution on the same grid.

(iii) ***Developers***:
Displays the information about developers in new window.

- Declared Functions:

  i) ***'get_grid'*** function:
  Type: Callback function of type void.
  Reads the 9x9 grid and stores the entries in the cells in the global variable and calls the solveSudoku function.

  ii) ***'menu_response'*** function :
  Type: Callback function of type void.
  Creates a menu bar in window and provides option like File, Help.

  iii) ***"sudoku_game"*** function:
  Type: Callback function of type void.
  Displays a window with 9x9 entry grid and a menu bar with certain number of cells in the grid filled, which cannot be changed by the user.

  iv) ***"developers"*** function:
  Type: Callback function of type void.
  Displays the information about the team members.

  vi) **"difficulty"** function:
  Type: Callback function of type void.
  Displays a window with options for the user to select different gameplay levels.

  vii) **"hint"** function:
  Type: Callback function of type void.
  Displays the hint in the next unfilled box of the 9X9 grid.

  viii) **"invalid_entry"** function:
  Type: bool.
  Checks and displays whether the Sudoku puzzle entered by the user is valid or not.

  ix) **"menu_about"** function:
  Type: Callback function of type void.
  Displays the rules of the game in a pop up window.

  x) **"Sudoku_display"** function:
  Type: Callback function of type void.
  Displays the 9X9 Sudoku game to the user.

  xi) **"check"** function:
  Type: Callback function of type void.
  Checks whether the values entered by the user so far are valid or not.

(**Reference:** http://www.gtk.org/)
www.youtube.com/user/the**cplusplusguy/**)

➢ Special mention:

In the SRS copy that we had submitted in stage-1, the logic of Sudoku generator function was decided as below:

▪ ‘sudoku_game’ Function:

The function performs the steps for generating a valid Sudoku puzzle and the working principles are given below:

1) Taking a 9X9 2-D array and initializing the values of all the 81 locations as zero.
2) Using a **rand** function to randomly choose a column number and a row number within the range 0 to 8.
3) Check if the location in the Sudoku grid with these row number and column number is empty (zero) or not.
    - ❖ If it is empty, then choose a random number in the range 1 to 9 and assign this number to the chosen location.
        - ➢ Then check if the number assigned in this location is safe or not using the **isSafetoPut** function. If it is safe then proceed to find the next empty location.
        - ➢ If the number assigned is not safe then change the number by another random number in the same range.
    - ❖ If it is not empty, then choose another location in the Sudoku grid by randomly choosing the row number and column number in the same range mentioned in the step 1.
4) Continue the above steps until 16 locations of the Sudoku grid is assigned by some numbers.
5) When it fills the 17th location by a number, it checks if the Sudoku puzzle has exactly one solution or not.
    - ❖ If it has a unique solution, proceed to find the next random location.
    - ❖ If it does not have a unique solution, then change the number in the location by another random number in the same range. It is guaranteed that the loop terminates because, when the Sudoku puzzle has a unique solution, the chosen location must have a number in the range 1 to 9.
6) According to the difficulty level chosen by the user, the function fills up certain number of random locations in the Sudoku grid.

- **Problem encountered:**

The above program was successfully built and executed, but the problem that we faced was that the time taken to generate a Sudoku puzzle was too long and the gameplay experience of the user would have definitely been unsatisfactory.

So we decided to go for the data file handling approach, which is illustrated in this stage-2 SRS document.